

PAT-NO: JP02001216140A

DOCUMENT-IDENTIFIER: JP 2001216140 A

TITLE: DEVICE FOR ALLOCATING INSTRUCTION
CACHE FUNCTION, METHOD FOR OPTIMIZING ALLOCATION AND RECORDING
MEDIUM RECORDING ALLOCATION OPTIMIZING PROCEDURE

PUBN-DATE: August 10, 2001

INVENTOR-INFORMATION:

NAME	COUNTRY
NIMATA, HIDEJI	N/A

ASSIGNEE-INFORMATION:

NAME	COUNTRY
NEC MICROSYSTEMS LTD	N/A

APPL-NO: JP2000027218

APPL-DATE: January 31, 2000

INT-CL (IPC): G06F009/06, G06F009/45 , G06F012/08

ABSTRACT:

PROBLEM TO BE SOLVED: To obtain an instruction cache function allocation device capable of reducing cache conflict between plural functions, and improving performing speed of an application program.

SOLUTION: A function time series information output part 100 inputs an application program 110 and outputs accessed function ID obtained at the time of accessing the function by a profile and returned function ID obtained at the time of return from the function to time series information

111 of function performance. A function time series information analysis part 101 analyzes the time series information 111, extracts a function accessing pattern having the possibility of generating direct/indirect cache conflict and records the extracted pattern as inter-function cache conflict combination information 112. A function memory space arrangement optimizing part 102 arranges a memory space in order from a function having the highest execution transition frequency so that a cache line is not shared.

COPYRIGHT: (C)2001,JPO

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-216140

(P2001-216140A)

(43) 公開日 平成13年8月10日 (2001.8.10)

(51) Int.Cl. ⁷	識別記号	F I	テーマコード [*] (参考)
G 0 6 F 9/06	4 1 0	G 0 6 F 9/06	4 1 0 H 5 B 0 0 5
9/45		12/08	S 5 B 0 7 6
12/08			W 5 B 0 8 1
		9/44	3 2 2 H

審査請求 有 請求項の数 8 O L (全 12 頁)

(21) 出願番号 特願2000-27218(P2000-27218)

(22) 出願日 平成12年1月31日 (2000.1.31)

(71) 出願人 000232036

エネイーシーマイクロシステム株式会社
神奈川県川崎市中原区小杉町1丁目403番
53

(72) 発明者 二俣 秀治

神奈川県川崎市中原区小杉町一丁目403番
53 日本電気アイシーマイコンシステム株
式会社内

(74) 代理人 100084250

弁理士 丸山 隆夫

Fターム(参考) 5B005 JJ13 LL04 MM02

5B076 AB02 BA01

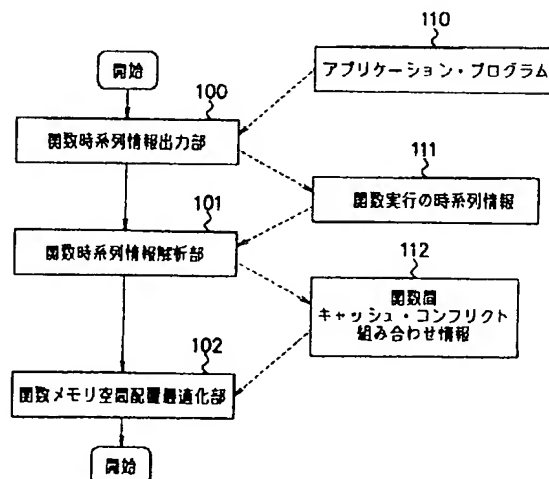
5B081 CC27

(54) 【発明の名称】 命令キャッシュ関数割付装置、割付最適化方法および割付最適化手順を記録した記録媒体

(57) 【要約】

【課題】 複数の関数間のキャッシュ・コンフリクトを削減し、アプリケーション・プログラムの実行スピードを向上する命令キャッシュ関数割付装置を得る。

【解決手段】 関数時系列情報出力部100がアプリケーション・プログラム110を入力してプロファイルにより関数呼び出し時の呼び出し先関数IDと関数からの戻り時の戻り先関数IDを関数実行の時系列情報111に出力し、この時系列情報111を関数時系列情報解析部101が解析し、直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出して関数間キャッシュ・コンフリクト組合せ情報112として記録する。関数メモリ空間配置最適化部102は、実行遷移回数の多い関数の順にキャッシュ・ラインを共有しないようにメモリ空間に配置する。



【特許請求の範囲】

【請求項1】 所定のアプリケーション・プログラムを入力してプロファイルにより関数呼び出し時の呼び出し先関数IDおよび該関数から戻り時の戻り先関数IDを出力する関数時系列情報出力部と、

前記関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、該抽出したパターンの関数の組合せと直接の関数呼出し関係にある関数の組合せおよびそれらの実行遷移回数を記録する関数時系列情報解析部と、

を有して構成されたことを特徴とする命令キャッシュ関数割付装置。

【請求項2】 前記関数時系列情報出力部から出力された呼び出し先関数IDと戻り先関数IDとを関数実行の時系列情報として蓄積する記憶部をさらに有することを特徴とする請求項1記載の命令キャッシュ関数割付装置。

【請求項3】 前記関数の組合せおよびそれらの実行遷移回数を関数間キャッシュ・コンフリクト組合せ情報として蓄積する記憶部を、さらに有することを特徴とする請求項1または2に記載の命令キャッシュ関数割付装置。

【請求項4】 プロファイルにより関数実行の時系列情報を出力し、

前記時系列情報から連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンの関数間キャッシュ・コンフリクト組合せ情報を検出し、

前記検出した関数間キャッシュ・コンフリクト組合せ情報に対して関数の中で複数の関数が連続して呼ばれている場合あるいはループの中で呼ばれている場合等のこれら複数の関数間のキャッシュ・コンフリクトを削減し、所定のアプリケーション・プログラムの実行スピードを向上したことを特徴とする命令キャッシュ関数割付最適化方法。

【請求項5】 前記関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、該抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を前記関数間キャッシュ・コンフリクト組合せ情報に記録することを特徴とする請求項4に記載の命令キャッシュ関数割付最適化方法。

【請求項6】 前記関数間キャッシュ・コンフリクト組合せ情報は、キャッシュ・コンフリクトを発生する可能性のある組合せの2つの関数IDを登録する2つの関数ID欄と該2つの組合せの関数実行遷移回数を設定する実行遷移回数欄とを備えていることを特徴とする請求項4または5に記載の命令キャッシュ関数割付最適化方

法。

【請求項7】 プロファイルにより関数実行の時系列情報を出力する手順と、

前記時系列情報から連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンの関数間キャッシュ・コンフリクト組合せ情報を検出する手順と、

前記検出した関数間キャッシュ・コンフリクト組合せ情報に対して関数の中で複数の関数が連続して呼ばれている場合あるいはループの中で呼ばれている場合等のこれら複数の関数間のキャッシュ・コンフリクトを削減する手順とを実行させるためのプログラムを記録した記録媒体。

【請求項8】 前記関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、該抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を前記関数間キャッシュ・コンフリクト組合せ情報に記録することを特徴とする請求項7に記載のプログラムを記録した記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、命令キャッシュへの関数割付けを最適化する命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体に関する。

【0002】

【従来の技術】従来、命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体は、例えば、CPUの速度に規制されつつ、メモリアクセス処理速度を改善する装置および方法に関する。これに関連する技術において、CPU速度とメモリアクセスの処理速度との差は、年々拡大する一方である。

【0003】DRAMのアクセス時間の向上率は年率7%、CPU速度は50～100%というデータもある

(1996年4月19日/日経BP社発行、日経BP出版センター発売の「コンピュータの構成と設計」、David A. Patterson、John L. Hennessy、成田光彰訳より)、よって、キャッシュの有効利用が、非常に重要な問題となってきた。なお、キャッシュとは既知の通り、外部メモリを高速にアクセスするための機構である。一般に、プログラムはメモリに配置された関数をアドレス順に処理していく。しかし、外部メモリのアクセス速度は、CPUの処理速度に比べて非常に遅い、このため、結果として実行速度が遅くなるという問題点がある。

【0004】上記の問題点を解消するため、関数実行の

際に外部メモリ上のプログラムをキャッシュという高速アクセス可能なバッファにコピーして実行することにより、高速なプログラム実行を実現することが可能となる。これは一般に、プログラムを実行すると、一度アクセスされたメモリは近いうちに再度アクセスされる可能性が高いという性質によるものである。

【0005】ただし、一般的にキャッシュの構成には、コストがかかるため外部メモリに比べて非常にサイズは小さい。よって、外部メモリをキャッシュのサイズで区切った領域に分割し、分割した領域毎に外部メモリをキャッシュに割り当て、あるアドレスに対する最初のアクセスでそのアドレスのプログラムをキャッシュの割り当てた領域にコピーし、次に外部メモリの同じアドレスをアクセスした場合は、キャッシュを直接アクセスする。このことで、高速なプログラム実行を実現する。

【0006】この時、外部メモリからキャッシュ・メモリへのコピーは特定のサイズの単位で行われ、このサイズでキャッシュを分割した領域をキャッシュラインと呼ぶ。従って、同一のキャッシュラインに割り当たった外部メモリのアドレスに配置された関数同士は、関数が切り替わる度にキャッシュにプログラムをコピーし直す必要が生じてくる。これをキャッシュ・コンフリクトという。これが頻発に起きると、結果としてプログラムの実行速度が遅くなってしまいう問題がある。よって、昨今では、この問題を解消すべく同時に動く可能性の高い関数同士は、同一のキャッシュラインには載らないように配置する方法が研究されている。

【0007】なお、従来、キャッシュには命令キャッシュとデータキャッシュがあるが、本発明は、命令キャッシュに着目するものである。外部メモリのキャッシュへの割当て方式には、最も単純で安価なダイレクト・マップ方式や、セット・アソシアティブ／フル・アソシアティブといった方式がある。しかし、基本的な問題は全て同じであるため、以降、ダイレクト・マップ方式を例にとり説明する。

【0008】また、従来の言語処理系プログラムでは、ある処理単位（関数）毎にメモリに適当に配置していた。このため、キャッシュ搭載のシステムにおいては、それが必ずしも最適に利用されているとは限らなかった。

【0009】昨今では、命令キャッシュに関し、キャッシュ・コンフリクトの回数を確率的に減らすために、関数の呼び出し回数情報を元にして関数のメモリ配置を最適化する。このことにより、キャッシュを有効に利用する研究がなされており、いくつかのアルゴリズムが論文発表されている。例えば、従来例1の“Efficient Procedure Mapping Using Cache Line Colorling” (A.H. Hemi, D.R. Kaeli, B. Calder ACM SIGPLAN, June, 1997) においては、呼び出し回数の多い関数から順にキャッシュ・コンフリクトを避けるように関数のメモリ配置を最適

にしていく手法が公開されている。

【0010】また、従来例2の特開平11-232117号公報の「プログラム変換方法、プログラム変換装置及びプログラム変換プログラムを記憶した記憶媒体」においては、キャッシュ・メモリを効率よく使用するための関数配置方法の実施例として上記従来例1の論文の方法が採用され、詳しく説明されている。この論文のアルゴリズムでは、関数の呼出グラフを作成し、呼び出し回数をその辺に対する重みとして優先順位を付けてメモリ空間に配置する。このことにより、まず関数を最初に配置した時のキャッシュ・コンフリクトを避けることができる。さらに、各関数が配置された「色」（＝使用されているキャッシュライン）と、その関数が現在利用できない「色」の集合を記録しておき、後者の色を使わないように関数を配置し、既に配置した関数についても、その関数の利用できない色を使わないという条件の下で、別の場所に移動する。このことで、直接の“親”あるいは“子”との間で発生するキャッシュ・コンフリクトを除去するものである。

【0011】図8、図9、および図10を参照し、図3に示す本発明の実施例で適用したソース・プログラムに対する従来例での動作を確認する。このことにより、従来例における問題点をより詳細に説明する。なお、図8および図9において、実線の矢印は処理の流れを示し、点線の矢印はデータの流れを示している。

【0012】図8の従来技術例に対し、図3に示すソース・プログラム例300を適用した場合、関数呼び出し情報出力部800において、プロファイルにより関数呼び出し時に呼出元と呼出先の関数情報とその呼出回数を出力すると、図10に示す関数呼出組合せ情報811が作成される。

【0013】次に、関数メモリ空間配置最適化部801において、関数呼出組合せ情報811を呼出回数の多い順にソートし、この順番にアドレス空間に配置すると同時に配置した関数が利用できない「色」の集合を認識し、これを避けて後続の関数を配置する。

【0014】すなわち、図9において、ステップ901では関数呼出組合せ情報811から図11に示す関数呼出グラフ1100を作成し、ステップ902では作成した関数呼出グラフ1100を呼び出し回数の多いものと少ないものに分割する。ここでは、func-funcA、func-funcB、func-funcCが前者の「多いもの」、main-funcが後者の「少ないもの」となる。ここで作成した呼出グラフにおいては、関数の組合せが辺となり、その両端のノードが組合せにおける2つの関数となる。なお、図3における各関数の占めるキャッシュライン数すなわち「色」の数は、funcが2個、funcA、funcB、funcC、mainが1個であるものとする。

【0015】次に、ステップ903において、呼出回数の多いもののグループを呼出回数の多い順にソートし、

その順番でステップ904以降の処理を行う。ステップ904では呼出回数の多い辺が残っているか確認し、残っているのでステップ905に進み、func-funcAの辺に対して両側のノードが未配置であるかを確認する。この確認において未配置であるのでステップ909に進み、funcとfuncAをメモリ空間上の任意の場所に隣接して配置し、ステップ915においてfuncとfuncAの利用できない「色」を利用不可能集合として認識した後、再びステップ904に戻る。隣接して配置されたfunc-funcAの辺は、複合ノードとして今後ひとつのノードとして扱われる。この時点で、既に配置済みの関数とキャッシュ・ラインの関係および各関数の利用不可能集合の状態は、図12a)に示す様になっている。

【0016】続いて、呼出回数の多い辺がまだ残っているのでステップ905に進み、func-funcCの辺に対して両側のノードが未配置であるかを確認する。この確認において、funcは配置であるのでステップ906に進み、2個の異なる複合ノードに属するノードを結ぶ辺かどうかを確認する。この確認において、funcCは複合ノードに属していないので、ステップ907に進み、一方のノードが複合ノードに属し他方のノードが未配置かどうか確認すると、条件に当てはまるのでステップ911に進む。

【0017】ステップ911では未配置のfuncCをfuncに近い場所に配置し、ステップ913において関数配置の際、利用不可能集合の影響で隙間が空いてないか確認すると空いていないので、ステップ915においてfuncとfuncCを利用できない「色」を利用不可能集合として認識した後に、再びステップ904に戻る。この時点で、既に配置済みの関数とキャッシュ・ラインの関係および各関数の利用不可能集合の状態は、図12b)に示す様になっている。

【0018】ステップ904において、未だ未配置の辺があるのでステップ905に進み、func-funcBの辺に対して両側のノードが未配置であるかを確認する。この確認において、funcは配置済みであるのでステップ906に進み、2個の異なる複合ノードに属するノードを結ぶ辺かどうかを確認する。この確認において、funcBは複合ノードに属していないのでステップ907に進み、一方のノードが複合ノードに属し、他方のノードが未配置かどうか確認すると条件に当てはまるのでステップ911に進む。

【0019】ステップ911では、未配置のfuncBと対を成すfuncの中心から複合ノードの両端までの距離が同じである。このため任意に左側に配置し、ステップ913において関数配置の際に利用不可能集合の影響で隙間が空いてないかを確認する。この認識の結果は空いていないので、ステップ915においてfuncとfuncBの利用できない「色」を利用不可能集合として認識した後、再びステップ904に戻る。ステップ904において、未

配置の辺が無くなったことを確認するとステップ916に進み、ただ1つの未配置ノードmainを、任意にfuncAの右に配置する。

【0020】図12c)は、最終的な関数配置とキャッシュ・ラインの関係、および各関数の利用不可能集合の状態であるが、funcA、funcBが同じキャッシュ・ライン「青」を共有しており、それぞれの利用不可能集合には「青」が含まれていない。よって、呼び出し元関数と呼び出し先関数の間のキャッシュ・コンフリクトは削減できる。

【0021】上記の従来例2では、手続き、関数、あるいはサブルーチン同士が、互いを呼び出す際のキャッシュ・メモリ上での衝突およびキャッシュ・ミス防止するのが目的である。この目的において、手続き、関数、あるいはサブルーチンが実際に呼び出される回数を示す情報と、手続き、関数、あるいはサブルーチン同士が、互いを呼び出す関係を示す情報とを利用している。これにより、手続き、関数、あるいはサブルーチン同士が、互いを呼び出す際のキャッシュ・メモリ上での衝突を防止できる。

【0022】

【発明が解決しようとする課題】しかしながら、上記の従来例2において、ある関数の中で複数の関数が連続して呼ばれている場合、あるいはループの中で呼ばれている場合等には、これら複数の関数間のキャッシュ・コンフリクトを削減できるとは限らないという問題点がある。

【0023】従来例2との比較において、本発明の目的は、互いの呼び出しに関係の無い手続き、関数あるいはサブルーチンがある手続き、関数あるいはサブルーチンの中で複数の手続き、関数あるいはサブルーチンが図3に示すプログラム例のfuncA、funcB、funcCのように連続して呼ばれている場合、あるいはループの中で呼ばれている場合に生じるキャッシュ・メモリ上での衝突およびキャッシュ・ミスを、防止することにある。

【0024】また、本発明の構成では、手続き、関数、サブルーチンの時系列情報を利用し、その時系列情報をパターン認識のアルゴリズムによる解析により抽出したキャッシュ・メモリ上での衝突およびキャッシュ・ミスを発生する可能性のある手続き、関数、あるいはサブルーチンの組合せ情報を利用している。この構成により、上記従来例2の効果に加え、互いの呼び出し関係に無い手続き、関数、あるいはサブルーチン同士のキャッシュ・メモリ上での衝突およびキャッシュ・ミスの発生をも、防止することができる。

【0025】本発明は、複数の関数間のキャッシュ・コンフリクトを削減し、アプリケーション・プログラムの実行スピードを向上する命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体を提供するこ

とを目的とする。

【0026】より詳細には、本発明は、プロファイルにより直接の関数呼出組合せ情報811を出力する替わりに、関数実行の時系列情報111を出力し、この時系列情報から、連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンを検出し、検出した関数間キャッシュ・コンフリクト組合せ情報112に対して、従来技術の関数配置最適化801を適用する。これにより従来削減できなかった、ある関数の中で複数の関数が連続して呼ばれている場合、あるいはループの中で呼ばれている場合等、これら複数の関数間のキャッシュ・コンフリクトを削減し、アプリケーション・プログラムの実行スピードを向上することを目的とする。

【0027】

【課題を解決するための手段】かかる目的を達成するため、請求項1記載の発明の命令キャッシュ関数割付装置は、所定のアプリケーション・プログラムを入力してプロファイルにより関数呼び出し時の呼び出し先関数IDおよび関数から戻り時の戻り先関数IDを出力する関数時系列情報出力部と、関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、この抽出したパターンの中の関数の組合せと直接の関数呼出し関係にある関数の組合せおよびそれらの実行遷移回数を記録する関数時系列情報解析部と、を有して構成されたことを特徴としている。

【0028】また、上記の関数時系列情報出力部から出力された呼び出し先関数IDと戻り先関数IDとを関数実行の時系列情報として蓄積する記憶部と、この関数の組合せおよびそれらの実行遷移回数を関数間キャッシュ・コンフリクト組合せ情報として蓄積する記憶部をさらに有するとよい。

【0029】請求項4記載の発明の命令キャッシュ関数割付最適化方法は、プロファイルにより関数実行の時系列情報を出力し、この時系列情報から連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンの中の関数間キャッシュ・コンフリクト組合せ情報を検出し、検出した関数間キャッシュ・コンフリクト組合せ情報に対して関数の中で複数の関数が連続して呼ばれている場合あるいはループの中で呼ばれている場合等のこれら複数の関数間のキャッシュ・コンフリクトを削減し、所定のアプリケーション・プログラムの実行スピードを向上したことを特徴としている。

【0030】また、上記の関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、この抽出したパターンの中の関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回

数を関数間キャッシュ・コンフリクト組合せ情報に記録し、さらに、この関数間キャッシュ・コンフリクト組合せ情報は、キャッシュ・コンフリクトを発生する可能性のある組合せの2つの関数IDを登録する2つの関数ID欄とこの2つの組合せの関数実行遷移回数を設定する実行遷移回数欄とを備えるるとよい。

【0031】請求項7に記載の発明の命令キャッシュ関数割付最適化手順を記録した記録媒体は、プロファイルにより関数実行の時系列情報を出力する手順と、時系列情報から連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンの中の関数間キャッシュ・コンフリクト組合せ情報を検出する手順と、検出した関数間キャッシュ・コンフリクト組合せ情報に対して関数の中で複数の関数が連続して呼ばれている場合あるいはループの中で呼ばれている場合等のこれら複数の関数間のキャッシュ・コンフリクトを削減する手順とを実行させることを特徴としている。

【0032】また、上記の関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、この抽出したパターンの中の関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を関数間キャッシュ・コンフリクト組合せ情報を記録するとよい。

【0033】

【発明の実施の形態】次に、添付図面を参照して本発明による命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体の実施の形態を詳細に説明する。図1から図7を参照すると、本発明の命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体の一実施形態が示されている。

【0034】本発明の特徴は、キャッシュを搭載したマイクロ・プロセッシング・デバイス・システム用アプリケーション・プログラムについて、プロファイルにより得られた関数単位の実行に関する時系列情報を解析する。特に、直接の呼び出し関係を持たない関数同士の実行において、キャッシュのコンフリクトを生じる可能性のある関数の実行パターンを検出し、これらの関数同士が実行時に同じキャッシュラインを共有しないようなメモリ空間上のアドレスに配置する。このことにより、プログラムの実行スピードを向上させる。この内容を以下に詳述する。

【0035】図1は、本発明の命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体の実施形態の処理手順例を示すブロック図である。また、図2は、関数間キャッシュ・コンフリクト組合せ情報112

の構成例を示す図である。なお、図1において、実線の矢印は処理の流れを示し、点線の矢印はデータの流れを示している。

【0036】関数時系列情報出力部100は、アプリケーション・プログラム110を入力し、プロファイルにより関数呼び出し時の呼び出し先関数IDと関数からの戻り時の戻り先関数IDを、関数実行の時系列情報111に出力する。

【0037】関数時系列情報解析部101は、関数実行の時系列情報111を解析し、直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、関数間キャッシュ・コンフリクト組合せ情報112に、抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を記録する。

【0038】関数メモリ空間配置最適化部102は、実行遷移回数の多い関数の順にキャッシュ・ラインを共有しないようにメモリ空間に配置する。また、実行遷移回数欄202は、関数ID欄200、201とその組合せの関数実行遷移回数を設定する。

【0039】関数間キャッシュ・コンフリクト組合せ情報112は、図2に示すように、キャッシュ・コンフリクトを発生する可能性のある組合せの2つの関数IDを登録する。なお、関数ID欄200、201とその組合せの関数実行遷移回数を設定する実行遷移回数欄202を備えている。

【0040】上記に構成される本実施形態の命令キャッシュ関数割付装置において、まず、関数時系列情報出力部100において入力したアプリケーション・プログラム110に対してプロファイルを行い、関数実行の時系列情報111を生成する。次に、関数時系列情報出力部100において生成された関数実行の時系列情報111を解析し、同じキャッシュ・ラインを共有していた場合に、実行においてキャッシュ・コンフリクトを生じる関数単位の実行遷移パターンを認識し、関数間キャッシュ・コンフリクト組合せ情報112に、認識された関数の組合せとその実行遷移回数を記録する。最後に、関数メモリ空間配置最適化部102において、関数間キャッシュ・コンフリクト組合せ情報112を参照し、実行遷移回数の多い関数の順に同じキャッシュ・ラインを共有しないようにメモリ空間に配置する。

【0041】以上の通り、プロファイルによる関数単位の実行に関する時系列情報を解析し、キャッシュのコンフリクトを生じる可能性のある関数の実行パターンを検出し、これらの関数同士が実行時に同じキャッシュ・ラインを共有しないようなメモリ空間上のアドレスに配置する。このことにより、プログラムの実行スピードを向上させることができる。これらの関係において、図1、図3、図4、図5、図6、および図7を参照して動作例について以下に詳述する。

【0042】図1において、アプリケーション・プログラム110を入力した関数時系列情報出力部100は、関数呼び出し時の呼び出し先関数IDと関数からの戻り時に戻り先関数IDを出力するコードを、アプリケーション・プログラム110に挿入して実行する。プロファイルと呼ぶこのことにより、これらの情報を関数実行の時系列情報111に出力する。

【0043】すなわち、アプリケーション・プログラム110のC言語によるソース・プログラムが図3に示すソース・プログラム例300の場合、関数funcの関数IDを“0”、関数funcAの関数IDを“1”、関数funcBの関数IDを“2”、関数funcCの関数IDを“3”、関数mainの関数IDを“4”と仮定する。本仮定において、まず、関数mainから関数funcの呼出時に呼出先関数IDの“0”を出力して、次に関数funcから関数funcAの呼出時に“1”を出力し、続いて関数funcAから関数funcへの戻り時に戻り先関数IDの“0”を出力し、以降、同様にしてプログラムの終了まで関数IDの列を出力する。

【0044】ソース・プログラム例300は、C言語で関数mainから関数funcを呼び出し、関数funcでは関数funcAと関数funcBを連続して呼び出す処理を10回繰返す処理と、続いて関数funcAと関数funcCを連続して呼び出す処理を20回繰返す処理を行い、関数mainに戻って終了するプログラムである。この処理により、プログラムの終了までプロファイルを実行すると、図4に示す関数IDの並びである関数時系列情報例400が作成される。

【0045】次に、関数時系列情報解析部101で、関数時系列情報出力部100で作成した関数実行の時系列情報111を解析し、直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、関数間キャッシュ・コンフリクト組合せ情報112に抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を記録する。

【0046】すなわち、図5を参照して関数時系列情報解析部101の詳細を説明する。なお、本図5において、実線の矢印は処理の流れを示し、点線の矢印はデータの流れを示している。まず、関数ID入力手段500において、関数時系列情報111から全ての関数IDの並びを入力する。この入力順に関数IDに“1”からの番号を付け、番号変数初期化処理501によって番号変数nを“1”に初期化する。

【0047】次に、関数時系列情報の終端判定処理502によってn+1が入力した関数IDの並びの総数を超えないと判定すると、再帰呼び出し判定処理503に進み、nとn+1の関数IDが“0”と“1”で異なるので再帰呼び出しではないと判定する。この判定の後、関数直接呼出し組合せ既登録判定処理504によってnと

n+1の関数IDの組合せが、関数間キャッシュ・コンフリクト組合せ情報112内に未だ存在しないと判定して、関数直接呼出し組合せ登録処理505によってnとn+1の関数ID(0, 1)の組合せを関数間キャッシュ・コンフリクト組合せ情報112に登録し、実行遷移回数欄に“1”を設定する。以上の処理によって、図6のa)に示す関数間キャッシュ・コンフリクト組合せ情報112が作成される。

【0048】さらに、認識パターン関数ID確認処理506によってn+4が関数IDの総数を超えないことを確認すると、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理507において、n、n+2、n+4の関数IDは“0”であり一致することを認識したと判定し、第二の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理508において、n+1、n+3の関数IDが“1”と“2”であり一致しないことを認識したと判定する。

【0049】次に、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理507の判定により、n+2はnと同一の関数IDであり、n+1とn+2は先の関数直接呼出し組合せ登録処理505によって登録済みのfuncとfuncAの組合せであることが自明となる。このため、関数戻り組合せ登録処理509によってn+1とn+2、すなわちnの関数ID(0, 1)の組合せの実行遷移回数に“1”を加算し、図6のb)に示す関数間キャッシュ・コンフリクト組合せ情報112が作成される。

【0050】続いて、コンフリクト関数組合せ既登録判定処理510によって、n+1とn+3の関数IDである“1”と“2”の組合せが関数間キャッシュ・コンフリクト組合せ情報112内に未だ存在しないと判定の後、コンフリクト関数組合せ登録処理511によって、n+1とn+3の関数ID(1, 2)の組合せを関数間キャッシュ・コンフリクト組合せ情報112に登録して実行遷移回数欄に1を設定し、さらに、次の関数時系列情報のパターン認識処理を行う。このため、番号更新処理512によって、nすなわち“0”をn+2すなわち“2”に更新し、関数時系列情報の終端判定処理502に戻る。この時点で、図6のc)に示す関数間キャッシュ・コンフリクト組合せ情報112が作成されている。

【0051】再び、関数時系列情報の終端判定処理502によってn+1が入力した関数IDの並びの総数を超えないと判定すると、再帰呼び出し判定処理503に進み、nとn+1の関数IDが“0”と“2”で異なるので再帰呼び出しではないと判定する。この後、関数直接呼出し組合せ既登録判定処理504によって、nとn+1の関数IDの組合せが関数間キャッシュ・コンフリクト組合せ情報112内に未だ存在しないと判定し、関数直接呼出し組合せ登録処理505によって、nとn+1の関数ID(0, 2)の組合せを関数間キャッシュ・コ

ンフリクト組合せ情報112に登録し、実行遷移回数欄に“1”を設定する。以上の処理によって、図6のd)に示す関数間キャッシュ・コンフリクト組合せ情報112が作成される。

【0052】さらに、認識パターン関数ID確認処理506によって、n+4が関数IDの総数を超えないことを確認すると、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理507において、n、n+2、n+4の関数IDは“0”であり一致することを認識したと判定し、第二の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理508において、n+1、n+3の関数IDが“2”と“1”であり一致しないことを認識したと判定する。

【0053】次に、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理507の判定により、n+2はnと同じ関数IDであり、n+1とn+2は先の関数直接呼出し組合せ登録処理505によって登録済みのfuncとfuncBの組合せであることが自明である。このため、関数戻り組合せ登録処理509によって、n+1とn+2、すなわちnの関数ID(1, 2)の組合せの実行遷移回数に“1”を加算する。

【0054】続いて、コンフリクト関数組合せ既登録判定処理510によって、n+1とn+3の関数IDである“2”と“1”の組合せが関数間キャッシュ・コンフリクト組合せ情報112に既に存在すると判定し、実行遷移回数更新処理515によって当該組合せの実行遷移回数に“1”を加算し、さらに、次の関数時系列情報のパターン認識処理を行う。このため、番号更新処理512によって、nすなわち“2”をn+2すなわち“4”に更新し、関数時系列情報の終端判定処理502に戻る。この時点で、図6のe)に示す関数間キャッシュ・コンフリクト組合せ情報112が作成されている。

【0055】以降、図4に示す関数IDの並びに対し、nが“40”になるまで同様の処理を繰り返すと、図6のf)に示す関数間キャッシュ・コンフリクト組合せ情報112が作成される。

【0056】さらに、図4に示す関数IDの並びである関数時系列情報例400に対して処理を進めると、この関数時系列情報例400の終端判定処理502によってn+1が関数IDの並びの総数を超えないと判定すると、再帰呼び出し判定処理503に進む。ステップ503の「nとn+1の関数IDは異なるか？」の判定において、nすなわち“40”とn+1すなわち“41”の関数IDが“0”と“1”で異なるので再呼び出しではないと判定し、関数直接呼出し組合せ既登録判定処理504によってnとn+1の関数IDの組合せが関数間キャッシュ・コンフリクト組合せ情報112内に既に存在すると判定して、実行遷移回数更新処理514によって当該組合せの実行遷移回数に“1”を加算する。

【0057】さらに、認識パターン関数ID確認処理5

06によって、 $n+4$ が関数IDの総数を超えないことを確認すると、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理507において、 n 、 $n+2$ 、 $n+4$ の関数IDが“0”であり一致することを認識したと判定し、第二の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理508において、 $n+1$ 、 $n+3$ の関数IDが“1”と“3”であり一致しないことを認識したと判定する。

【0058】次に、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理507の判定により、 $n+2$ は n と同一の関数IDであり、 $n+1$ と $n+2$ は先の関数直接呼出し組合せ登録処理505によって、登録済みのfuncとfuncAの組合せであることが自明となる。このため、関数戻り組合せ登録処理509によって $n+1$ と $n+2$ 、すなわち n の関数ID(1, 3)の組合せの実行遷移回数に“1”を加算する。

【0059】続いて、コンフリクト関数組合せ既登録判定処理510によって、 $n+1$ と $n+3$ の関数IDである“1”と“3”の組合せが関数間キャッシュ・コンフリクト組合せ情報112にまだ存在しないと判定し、コンフリクト関数組合せ登録処理511によって $n+1$ と $n+3$ の関数IDの組合せを関数間キャッシュ・コンフリクト組合せ情報112に登録し、実行遷移回数欄に“1”を設定する。さらに、次の関数時系列情報のパターン認識処理を行うため、番号更新処理512によって n すなわち“40”を $n+2$ すなわち“42”に更新し、関数時系列情報の終端判定処理502に戻る。この時点で、図6のg)に示す関数間キャッシュ・コンフリクト組合せ情報112が作成される。

【0060】再び、関数時系列情報の終端判定処理502によって、 $n+1$ が入力した関数IDの並びの総数を超えないと判定すると再帰呼び出し判定処理503に進み、 n と $n+1$ の関数IDが“0”と“3”で異なるので再起呼び出しではないと判定する。すると、関数直接呼出し組合せ既登録判定処理504によって、 n と $n+1$ の関数IDの組合せが関数間キャッシュ・コンフリクト組合せ情報112内にまだ存在しないと判定して、関数直接呼出し組合せ登録処理505によって、 n と $n+1$ の関数ID(0, 3)の組合せを関数間キャッシュ・コンフリクト組合せ情報112に登録し、実行遷移回数欄に“1”を設定する。

【0061】さらに、認識パターン関数ID確認処理506によって、 $n+4$ が関数IDの総数を超えないことを確認すると、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理507において、 n 、 $n+2$ 、 $n+4$ の関数IDが“0”であり一致することを認識したと判定する。この後、第二の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理508において、 $n+1$ 、 $n+3$ の関数IDが“3”と“1”であり、一致しないことを認識したと判定す

る。

【0062】次に、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理507の判定により、 $n+2$ は n と同じ関数IDであり、 $n+1$ と $n+2$ は先の関数直接呼出し組合せ登録処理505によって登録済みのfuncとfuncCの組合せであることが自明である。このため、関数戻り組合せ登録処理509によって、 $n+1$ と $n+2$ 、すなわち n の関数ID(0, 3)の組合せの実行遷移回数に“1”を加算する。

【0063】続いて、コンフリクト関数組合せ既登録判定処理510によって、 $n+1$ と $n+3$ の関数IDである“3”と“1”の組合せが関数間キャッシュ・コンフリクト組合せ情報112内に既に存在すると判定し、実行遷移回数更新処理515によって当該組合せの実行遷移回数に“1”を加算する。さらに、次の関数時系列情報がパターンに合致するか確認するため、番号更新処理510によって、 n すなわち“42”を $n+2$ すなわち“44”に更新し、関数時系列情報の終端判定処理502に戻る。この時点で、図6のh)に示す関数間キャッシュ・コンフリクト組合せ情報112が作成されている。

【0064】以降、図4に示す関数IDの並びである関数時系列情報例400に対し、 n が“120”になるまで同様の処理を繰り返すと、図6のi)に示す関数間キャッシュ・コンフリクト組合せ情報112が作成される。

【0065】さらに、図4に示す関数IDの並びである関数時系列情報例400に対し、処理を進めると、関数時系列情報の終端判定処理502によって $n+1$ が関数IDの並びの総数を超えないと判定する。すると、再帰呼び出し判定処理503に進み、 n と $n+1$ の関数IDが“0”と“4”で異なるので再起呼出ではないと判定し、関数直接呼出し組合せ既登録判定処理504によって、 n と $n+1$ の関数IDの組合せが関数間キャッシュ・コンフリクト組合せ情報112内にまだ存在しないと判定する。この後、関数直接呼出し組合せ登録処理505によって、 n と $n+1$ の関数ID(0, 4)の組合せを関数間キャッシュ・コンフリクト組合せ情報112に登録し、実行遷移回数欄に“1”を設定する。

【0066】さらに、認識パターン関数ID確認処理506によって、 $n+4$ が“124”であり、関数IDの総数121を超えていることを確認すると、番号更新処理513に進み、 n すなわち“120”を $n+1$ すなわち“121”に更新して関数時系列情報の終端判定処理502に戻り、 $n+1$ が関数IDの総数を超えたと判定され、関数時系列情報解析部101の処理を終了する。この時点で、図6のj)に示す関数間キャッシュ・コンフリクト組合せ情報112が、作成されている。

【0067】最後に、関数メモリ空間配置最適化部103で、図9に示す従来技術と同じ処理を、従来の関数組

合せ情報811の代わりに関数間キャッシュ・コンフリクト組合せ情報112を入力として行う。この後、関数間キャッシュ・コンフリクト組合せ情報にfuncAとfuncBの組合せが含まれることにより、図7に示す最終的な関数配置とキャッシュ・ラインの関係、および各関数の利用不可能集合の状態にみられるように、funcA、funcBが同じキャッシュ・ラインを共有すること無く配置される。

【0068】尚、上述の実施形態は本発明の好適な実施の一例である。但し、これに限定されるものではなく、本発明の要旨を逸脱しない範囲内において種々変形実施が可能である。

【0069】

【発明の効果】以上の説明より明らかなように、本発明の命令キャッシュ関数割付装置では、関数時系列情報から間接的キャッシュ・コンフリクトを発生する可能性のある関数の呼び出しパターンを抽出し、直接の関数呼び出しによるキャッシュ・コンフリクトを発生する可能性のある関数呼び出しパターンと合わせて、その関数実行遷移回数の多い順に同じキャッシュ・ラインを共有しないようにメモリ空間上に配置する。この直接、間接的なキャッシュ・コンフリクトを極力発生しないようにメモリ空間上に配置することにより、アプリケーション・プログラムの実行スピードを向上することができる。

【図面の簡単な説明】

【図1】本発明の命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体の実施形態の処理手順例を示すブロック図である。

【図2】関数間キャッシュ・コンフリクト組合せ情報112の構成例を示す図である。

【図3】ソース・プログラム例300を示す図である。

【図4】関数時系列情報例400を示す図である。

【図5】関数時系列情報解析部101の処理手順例を示すフローチャートである。

【図6】関数間キャッシュ・コンフリクト組合せ情報112を示す図である。

【図7】最終的な関数配置とキャッシュ・ラインの関係、および各関数の利用不可能集合の状態を示す図である。

【図8】従来の命令キャッシュ関数割付装置の処理手順例を示すブロック図である。

10 【図9】従来の関数メモリ空間配置最適化部801の処理手順例を示すフローチャートである。

【図10】従来の関数呼出組合せ情報811の構成例を示す図である。

【図11】従来の関数呼出グラフ1100の構成を説明するための図である。

【図12】従来の関数配置とキャッシュ・ラインの関係および各関数の利用不可能集合の状態を示す図である。

【符号の説明】

100 関数時系列情報出力部

20 101 関数時系列情報解析部

102 関数メモリ空間配置最適化部

103 関数メモリ空間配置最適化部

110 アプリケーション・プログラム

111 関数実行の時系列情報

112 関数間キャッシュ・コンフリクト組合せ情報

200、201 関数ID欄

202 実行遷移回数欄

300 ソース・プログラム例

500 関数ID入力手段

501 番号変数初期化処理

502 関数時系列情報の終端判定処理

503 再帰呼び出し判定処理

504 関数直接呼び出し組合せ既登録判定処理

【図2】

112

関数ID欄	実行遷移回数
200	202
201	

【図3】

300

```

void func 0 .....0
{
    int i;
    for (i=0; i<10; i++) {
        funcA 0 .....1
        funcB 0 .....2
    }
    for (i=0; i<20; i++) {
        funcA 0 .....1
        funcC 0 .....3
    }
}

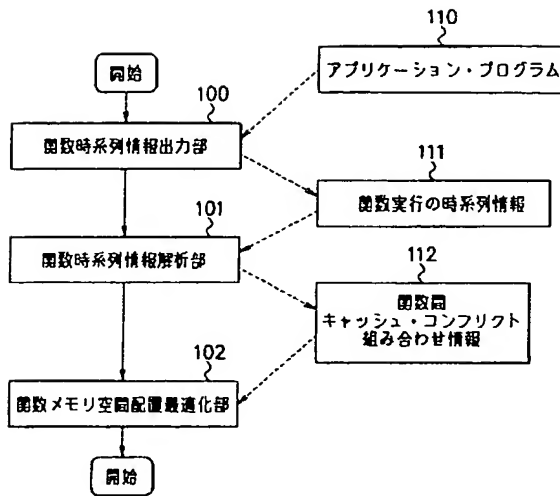
void main 0 .....4
{
    func 0; .....0
}

```

【図10】

呼出し元	呼出し先	実行遷移回数
main	func	1
func	funcA	30
func	funcB	10
func	funcC	20

【図1】

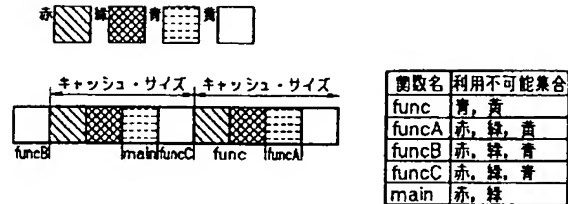


【図4】

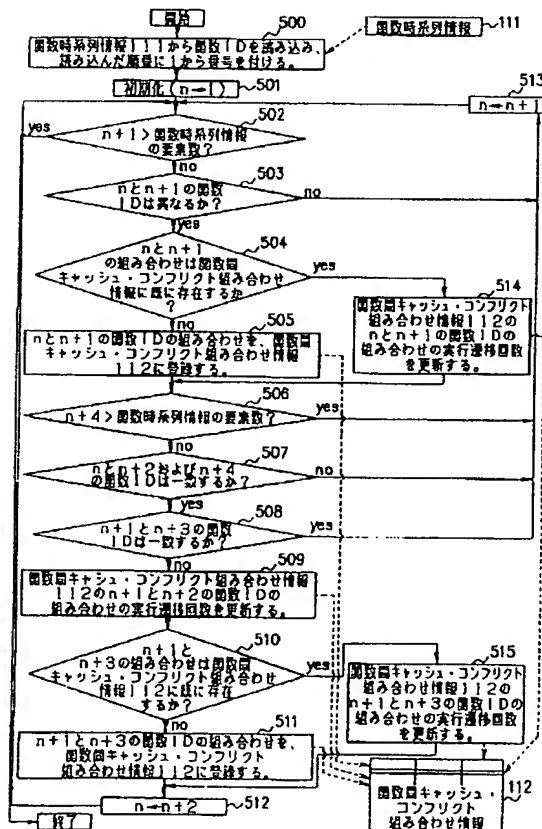
400

0, 1, 0, 2, (以降同じ並びの9回繰り返し), 一行に続く
0, 1, 0, 3, (以降同じ並びの9回繰り返し), 4

【図7】



【図5】



【図6】

a)

組み合わせ	実行遷移回数
func (0) funcA (1)	1

b)

組み合わせ	実行遷移回数
func funcA (1)	2

c)

組み合わせ	実行遷移回数
func (0) funcA (1)	2
funcA (1) funcB (2)	1

d)

組み合わせ	実行遷移回数
func (0) funcA (1)	2
funcA (1) funcB (2)	1
func (0) funcB (2)	1

e)

組み合わせ	実行遷移回数
func (0) funcA (1)	2
funcA (1) funcB (2)	2
func (0) funcB (2)	2

f)

組み合わせ	実行遷移回数
func (0) funcA (1)	20
funcA (1) funcB (2)	20
func (0) funcB (2)	20

g)

組み合わせ	実行遷移回数
func (0) funcA (1)	21
funcA (1) funcB (2)	20
func (0) funcB (2)	20
funcA (1) funcC (3)	1

h)

組み合わせ	実行遷移回数
func (0) funcA (1)	21
funcA (1) funcB (2)	20
func (0) funcB (2)	20
funcA (1) funcC (3)	2
func (0) funcC (3)	2

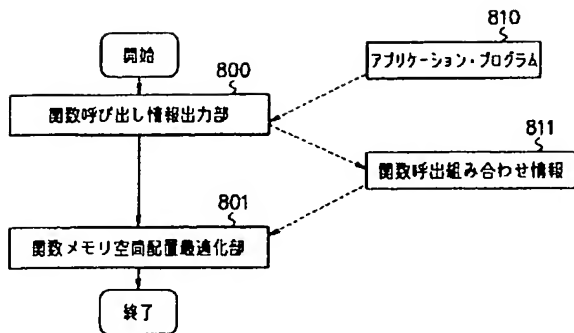
i)

組み合わせ	実行遷移回数
func (0) funcA (1)	60
funcA (1) funcB (2)	20
func (0) funcB (2)	20
funcA (1) funcC (3)	38
func (0) funcC (3)	39

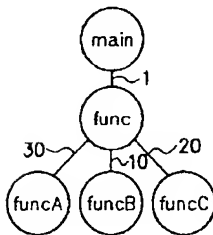
j)

組み合わせ	実行遷移回数
func (0) funcA (1)	60
funcA (1) funcB (2)	20
func (0) funcB (2)	20
funcA (1) funcC (3)	38
func (0) funcC (3)	39
func (0) main (4)	1

【図8】

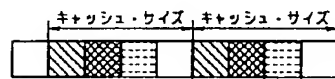


【図11】



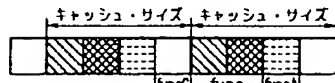
【図12】

a)



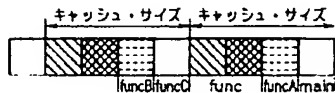
関数名	利用不可能集合
func	青, 黄
funcA	赤, 緑
funcB	未配置
funcC	未配置
main	未配置

b)



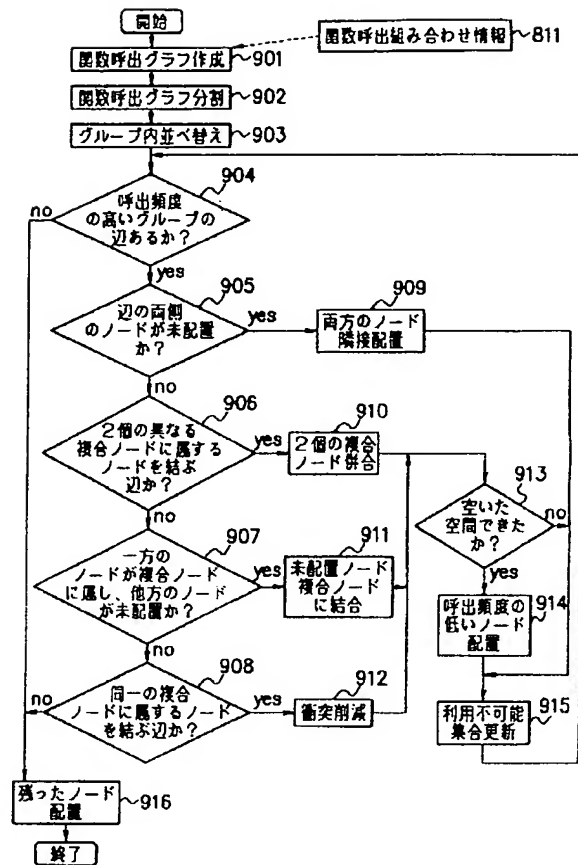
関数名	利用不可能集合
func	青, 黄
funcA	赤, 緑
funcB	未配置
funcC	赤, 緑
main	未配置

c)



関数名	利用不可能集合
func	青, 黄
funcA	赤, 緑
funcB	赤, 緑
funcC	赤, 緑
main	赤, 緑

【図9】



【手続補正書】

【提出日】平成 12 年 2 月 25 日 (2000. 2. 25)

【手続補正 1】

【補正対象書類名】明細書

【補正対象項目名】発明の名称

【補正方法】変更

【補正内容】

【発明の名称】 命令キャッシュ関数割付装置、割付最適化方法および割付最適化手順を記録した記録媒体